OUTPQX Commentary

Queue message to network

Tue, Feb 6, 2001

```
Function OUTPQX(block: BlockPtrType; used: Boolean): Boolean;
```

This note describes the logic used by the routine OUTPQX, which is used by the system software for queuing any network message. The `block` argument is a pointer to a message block, of which many types are used by the system. The `used` argument is a Boolean flag for which "true" means place the message block into the appropriate network output pointer queue, but mark it so that it will not actually be sent to the network hardware. (One use for this option is passing an alarm message block to a network output message queue so that it can later QMonitor Task processing can encode a readable alarm message for serial port output. QMonitor monitors messages that pass through all output pointer queues, and it can optionally generate such messages.) In most cases, the `used` argument will be specified as false, or zero.

For every message block, there is a word field included that gives the offset (from the start of the block) to the word containing the destination node# for the message. The location of this offset word fits one of two cases. For block type# 2, this offset word is found at offset 20 (decimal) bytes. For all other message block types, the offset word is found at offset 2 bytes. So, to find the destination node for a message block, one uses as an offset the word found at one of these two offset locations.

Using the above rule, access the destination node# for the message contained within the block. If it is zero, call BCNODE to get the "broadcast node#" in use by this station to use instead. Thus, a zero destination node# value is shorthand for the "broadcast node#," which normally specifies a multicast target address. (This feature is probably of dubious benefit and is seldom, if ever, used.)

The code next examines the destination node# to determine the network to be used. By default, the network is token ring. But in several possible cases, ethernet is used instead. If the node# has the sign bit set, use ethernet. For a 162 board CPU, if the node# is in the range `0x09xx` or in the range `0x6xxx`, or if no token ring interface board is present, use ethernet. If the node# is in the range `0x7Axx`, use arcnet.

If the node# is in the range `0x05xx—0x07xx`, and the "broadcast node# is in the range `0x09xx`, and the message is an Acnet protocol message that is *not a reply* message, then try to use IP by invoking IPNodeN to get a pseudo node# (range `0x6xxx`). (This depends upon access to the IPNAT cache to find an IP address of the target node#. (IPNAT is updated by the DNSQ local application that queries the Domain Name Server to obtain IP addresses for node names such as node0562.) If successful, meaning that the IP address for this given destination node# was already contained within the IPNAT cache so that a pseudo node number can be returned, then use it.

For any Acnet message targeted via a pseudo node#, call FindUDP to obtain the target IP address from the IPARP table and search the Acnet trunk tables to get an Acnet node# to use as the target node# instead. On a 162 board, always use ethernet for IP communications.

Assuming that the "used" argument is false, if the target node# is now an Acnet UDP node#, call UNodeChk to lookup the IP address from the Trunk tables and get a pseudo node# via PsNIPARP that specifies either the Classic or Acnet port#. UNodeChk also installs the Acnet UDP node# in the NODEN field of the IPARP entry.

If the target node# is now a pseudo node#, determine the network number being used. Call CheckARP to test whether an ARP request is needed, based upon whether a physical node address currently exists in the IPARP entry. If an ARP request is needed, CheckARP queues a pointer to this message block onto an ARP queue associated with this node's IPARP entry, and it also queues the ARP request. (It calls OUTPQX to do this, but that's ok.) This scheme allows building up a queue of message blocks for later processing upon arrival of the ARP reply. As soon as the ARP reply arrives, these queued message blocks will again be submitted to OUTPQX. In the meantime, any attempt to send more messages to the same target will not result in actually placing an entry into the appropriate output pointer queue.

Assuming that everything is ok to this point, it is time to install a new entry into the output pointer queue. If the "used" argument is nonzero, or true, then set $80 into the USEDELAY byte, else clear that byte. If the sign bit is set in this byte, the message will not actually be sent to a network. The rest of the byte is used to time out messages waiting in the queue for a given network. Place the block type# into the BLOCKTYP byte of the OUTPQ entry. Place the final derived node# into the DESTNOD field. Advance the IN offset word of the queue. If the queue is not full, assign ownership of the message block to the QMonitor task, and return true value, else do not advance IN, and return false.

Examples

The above description of OUTPQX flow may be less than transparent. Consider a few examples taken from real life. First, assume that we have an ethernet-based IRM, and the message to be sent is a reply message for some data request, using either Classic or Acnet protocol. In this case, the destination node# will be of the form `0xExxy`. For this case, the important parts of the flow are merely the call to CheckARP to insure that an ARP request will not be needed, followed by actually installing the OUTPQX entry. This simple path for replies to data requests is likely to be by far the most common one.

As a second example, assume a Classic data request is sent by one IRM to another IRM, and that the destination node is of the form `0x05xx`. Ethernet will be used. IPNodeN will be called to obtain a pseudo node# based upon an IP address found in the IPNAT. Assuming this is successful, CheckARP is called to be sure an ARP request is not needed, and if successful, the OUTPQ entry is installed using the pseudo node#.

A third example is one of initiating an Acnet request to an Acnet node# such as `0x0987`. In this case, UNodeChk is called to obtain a pseudo node# based upon Trunk table lookup. Then CheckARP is called, which if successful, results in installing the entry.

A fourth example is an Acnet request to another IRM node using its native node#, such as `0x0562`. In this case, IPNodeN is called to find a matching IPNAT entry to get the IP address and convert to a pseudo node#. Then FindUDP is called to search the Trunk tables to find the target node's Acnet node# such as `0x09C0`. Then UDodeChk is called to again convert to a pseudo node# using the Trunk table. Finally CheckARP is called and the entry is installed. This may not be an optimized case, but it is unusual in practice.

All the above examples use IP, although support for raw network communications is still included in the system code. With the passing of token ring, there will no longer be any need for continuing to support raw network communications, as everything will be IP-based.